The objective of this lab is to become familiar with using UDDI services to dynamically publish, discover, and integrate web services programmatically from your application.

In this lab, you will extend both the book store web service and the client applications you built as part of Lab 4. Your web service will support additional methods that will control whether or not information about the web service is available in the local UDDI server. At run time, you will run multiple instances of this service (corresponding to different URL bindings). Your client programs will now satisfy each request by first contacting the UDDI server to discover the service instances that are available, make method invocations on these services, and present the merged results to the user.

Preliminaries:

A. Compiling and Running the Lab5 Starter Application

The discussion below refers to the starter files I have made available as a zipped-folder Lab5zip on each of the development servers at the following location:

D:\VSDev\Public\vijayk\Lab5.zip

Please expand this folder into the following location: D:\VSDev\<your user name>\Lab5.

This folder expands into a Visual Studio.NET solution, which contains one project: SearchClient. Load the Lab5 solution into Visual Studio.NET. Build the solution and run the program with zero, one, and two integer arguments. In each case, you should see writes to console that refer to two services that I have inserted into the UDDI server on each machine.

B. Web-based Interaction with the UDDI Service

The UDDI Service on each machine is accessible at the following URL: http://localhost/uddi.

When you browse to this URL, you should see a page, which identifies you as a “Publisher” on the top right (please let me know if this is not the case). You will also see a banner area with three options: “Home”, “Search”, and “Publish”. These words are clickable, and take you to the current page, the search interface, and the publish interface respectively.

1. Become familiar with the different kinds of searches that you can make against the UDDI server. Note that the character “%” denotes a wildcard. You can match against all providers by supplying “%” in the provider field.

2. Click on the “Publish” option. You will be shown a page where you can add “Providers” and “tModels”. Note that you will be only shown information that you have published (the first time around, you should see no “Providers” or “tModels” previously published).

Following the structure of the entries I have already inserted into the UDDI server, add a provider entry for yourself. Add a tModel for the WSDL file corresponding to one of the sample services that were supplied as part of Lab 4. Identify the “Category” of the tModel as following the categorization scheme (uddi-org:types). Follow the category hierarchy till you find one that says “Specification for a web service described in WSDL” and click on it. If done correctly, this should identify your tModel as having a key value “wsdlSpec”.

1
Once you have added both the provider and the tModel, you can now add a “Service” under that provider, and a “Binding” for that “Service”. Part of the “Binding” record requires you to specify the URL where the service is accessible (the “Access Point”): provide the URL where the service your are adding is present. Associate the tModel you added earlier with the “InstanceInfo” associated with this “Binding”.

Verify that you did the above steps correctly by conducting a “Search” against the UDDI service for your provider name, the tModel that you added, etc.

3. Create a new instance of the web service from Step 2 as follows. In Visual Studio.NET, create a new C# project of type: “ASP.NET Web Service”, and give it the location where you want the new service to reside. Save the solution and exit Visual Studio.NET. Go to the directory that corresponds to the URL that you specified, and delete its contents (leave the directory intact). Copy the following files from the directory that contains the original service from Step 2: *.asmx, *.asax, *.config, and bin/*.dll into this new directory (you should have a bin subdirectory).

Verify that you did the above correctly by browsing to the service URL: you should see information about the service methods.

4. Follow the instructions in Step 2 for publishing this new service to the UDDI Server. Note that you should not create a new tModel entry. Since the new service you have created is a copy of the first one, the same WSDL file should suffice to describe it. Note that you will not be using the access point encoded within the WSDL file, but instead will replace it with the Access Point information obtained from UDDI.

5. Modify the UDDIInteraction project so that it refers to your provider name and your WSDL file name (instead of “vijayk” and “SumAndDifferenceWSDL-vijayk”). Rebuild and rerun it: you should now see information about the services you have added.

C. Using the Microsoft.Uddi Assembly

To interact with the UDDI server programmatically, you will be using classes defined in the Microsoft.Uddi assembly. For each project that interacts with UDDI, you will need to add a reference to the above assembly (you will find it in the list of .NET assemblies).

Additional information about the functions in the UDDI assembly (including source code samples) can be found online at the following URL:


Unfortunately, this documentation is too new to make it into the MSDN library we have currently installed on the netserver* machines (which is accessible from Visual Studio.NET).

What You Need To Do:

1. Make sure that you have completed each step described in the “Web-based Interaction with the UDDI Service” section above.

2. Add three new Visual C# projects to the Lab5 solution: (i) an “ASP.NET Web Service” called MyBookService corresponding to the following URL: http://localhost/VSDev/<your user name>/Lab5/WebServices/MyBookService; (ii) a “Console Application” called MyBookServiceAdmin; and (iii) an “ASP.NET Web Application” called InteractionForm. Save and close the solution.

Delete the default contents inserted by Visual Studio.NET into the three projects above, and copy in the contents of the corresponding directories from your (completed) Lab 4 solution. **Make sure that you copy the files, not just move them.** Update your web references so that they refer to Lab 5’s services.
Reload the Lab5 solution, rebuild, and verify that your Lab 4 solution behaves as expected (even if the files are now part of the Lab5 solution).


4. Extend the functionality of the MyBookService web service so that it supports two additional methods:

   string Publish()
   Should programmatically interact with the UDDI publish interface to add a “Service” record corresponding to this service (determine the service name by parsing the URL associated with the current request: “Context.Request.Url” gives you this information). The owner of this service should be the provider you set up as part of the web-based interaction with the UDDI service. Once the service record is added, add a “Binding” record for that service, which refers to the URL that this service is available at. The “InstanceInfo” field for the binding record should refer to the tModel that you will establish in Step 7 below (before running the client applications): decide on what the name for that tModel will be, and use that name in your code.

   string Unpublish()
   Should delete the “Service” and “Binding” entries you published using Publish().

   Note that you should keep track of whether or not information about the service is currently published: calling Publish on a service that has already Published its information, or calling Unpublish on a service for which no information is currently published should both return with explanatory messages.

5. Extend the functionality of the MyBookServiceAdmin application so that you can invoke the Publish and Unpublish methods on your web service. Verify that you have implemented the required functionality correctly by interacting with the UDDI server using the web interface (searching for the service should find it only during the period where the service information is published).

   Since you will have to use this application to interact with multiple instances of the web service, modify it so that it takes a command-line argument specifying the URL of the service instance that this application will be connecting to. Note that each of the instances have the same WSDL, so you can leave the web reference as is, and just modify the URL of the proxy object you create in your program as follows: (assumes that the web reference uses the name “MyBookService”)

   ```csharp
   MyBookService proxy1 = new MyBookService();
   proxy1.Url = <URL of the service instance you are connecting to>;
   ```

6. Extend the functionality of the InteractionForm application so that before answering each search query, it first contacts the UDDI server for information about currently published services and then invokes the search query on them. You should first invoke the SearchISBNs method on each of the discovered services, merge the records (i.e., drop duplicates), and then issue one or more GetRecordsHTML invocations. You can make this procedure more fancy by first sorting the returned ISBNs and then retrieving the HTML corresponding to the sorted list.

   When searching for services, please limit your search to only those services that you have published (i.e., corresponding to your provider), and those that implement the tModel that you specify in Step 7 below. For each of the services, you will need to retrieve information about their access point and then create a proxy for talking to the service at that access point.

   In order to do this, you should (statically) set up a web reference to the WSDL file corresponding to your updated MyBookService web service (note that although at run time, your program will be interacting with multiple instances of this service, they all correspond to the same WSDL). This allows you to create a proxy for each service instance in your code as in the code fragment shown above.
7. Using the Web-based interface add into the UDDI server a tModel corresponding to the WSDL of your extended MyBookService from Step 4. Note that if you use a URL that refers to the dynamically generated WSDL ( `<service name>?WSDL` ) you can set this up once at the beginning of this whole exercise and leave it untouched after that (even as you are debugging the service in Step 4).

8. Once you are happy that your MyBookService implementation works well, copy its `*.asmx`, `*.asax`, `*.config`, and `bin/*.dll` files into the instance directories you have set up in Step 3. Verify that you did get all the files by browsing to the service locations. At this point, you should have four instances of the service corresponding to MyBookService, MyBookService1, MyBookService2, and MyBookService3.

9. Start the MyBookServiceAdmin program from four command prompt windows, supplying as command-line arguments different service URLs. Start the InteractionForm from a different command prompt. You should be able to exercise the whole application as follows: populate each of the services with a different (possibly overlapping) range of ISBNs; update prices as you desire, and perform a bunch of keyword/price searches while invoking the Publish and Unpublish methods on the services (from their admin programs). Your searches should return the expected results (merged from among the services active at the time of the request).

10. Handle errors and exceptions as gracefully as you can (e.g., by catching the appropriate exception and printing out an error message or informing the user about the error via appropriately generated HTML.

Submit the lab folder information and writeup using the web-based submission form on the development server where you have your account. If you follow the guidelines above, the folder should just be: `D:\VSDev\<your user name>\Lab5`.